FedCod: An Efficient Coded Communication Protocol for Cross-Silo Federated Learning

Peishen Yan[†], Jun Li[‡], Hao Wang[§], Yang Hua[¶], Tao Song[†], Lu Peng[‖], Haibing Guan[†*]

[†]Shanghai Jiao Tong University, [‡]City University of New York, [§]Stevens Institute of Technology,
[¶]Queen's University Belfast, [‖]Tulane University
{peishenyan, songt333, hbguan}@sjtu.edu.cn, jun.li@qc.cuny.edu,
hwang9@stevens.edu, y.hua@qub.ac.uk, lpeng3@tulane.edu

***Abstract*—Federated Learning (FL) is an innovative distributed machine learning paradigm that enables multiple parties to collaboratively train a model while preserving data privacy. Communication efficiency concerns arise in cross-silo FL, particularly due to the network heterogeneity and fluctuations associated with geo-distributed data silos. Most of the existing solutions to these problems focus on algorithmic improvements that alter the FL algorithm but sacrifice the training performance. How to address these problems from a network perspective that is decoupled from the FL algorithm remains an open challenge. This paper proposes FedCod, the first application-layer coded communication protocol designed for cross-silo FL. FedCod transparently utilizes a coding mechanism to enhance the efficient use of idle bandwidth through client-to-client communication, and dynamically adjusts coding redundancy to mitigate network bottlenecks and fluctuations, thereby improving the communication efficiency and accelerating the FL training process. In our real-world experiments, FedCod demonstrates a significant reduction in average communication time by up to 62% compared to the baseline, while maintaining FL training performance and optimizing inter-client communication traffic.**

***Index Terms*—Federated learning, coded communication, geographically distributed data centers.**

I. Introduction

Unlike cross-device federated learning (FL) that distributes model training to loosely connected devices (*e.g.*, Internet-of-Things devices and smart phones) [1], [2], cross-silo FL performs distributed training across geo-distributed data centers (known as "silos") powered with sufficient computing resources, connected by wide-area networks (WANs), and provisioned with large amounts of data [3], [4]. Cross-silo FL has been widely applied to enable collaborative training between organizations to build artificial intelligence (AI) models without sharing their private data, such as healthcare companies [5], [6], [7] and financial institutions [8]. The release of the EU AI Act [9], the popularity of large language models (LLMs), and scarcity of high-quality data for LLM training will jointly boost the development and deployment of cross-silo FL [10], [11].

However, cross-silo FL has been suffering from real-world network conditions, leading to inflated training time and inefficient utilization of client data: **1) Increasing bandwidth demand**: With explosively increasing neural network model sizes [12], [13], particularly LLMs [14], the limited WAN bandwidth has been throttling cross-silo FL's training efficiency [15]. **2) Network heterogeneity**: The geo-distributed nature of cross-silo FL leads to heterogeneous connections between silos, creating bottlenecks that extensively slow down the exchange of model weights [15], [16], [17]. **3) Bandwidth fluctuations**: Variability in WAN traffic over public links leads to fluctuating bandwidth over time [18], [19], exacerbating training inefficiencies by increasing transmission delays.

Most existing methods address these problems through algorithmic innovation, either by performing model compression [20], [21] or by refactoring training and aggregation strategies [22], [23], [24]. However, such algorithmic solutions typically trade off between model quality and training efficiency, which may result in sub-optimal performance under highly volatile cross-silo FL networks [25], [26], [27], [28]. This dilemma inspires researchers to further explore solutions from a network perspective [29], [30], [31]. For example, HierFL [30] organizes clients into clusters and performs hierarchical model aggregation based on network topology and geographic proximity to reduce bandwidth consumption and improve communication efficiency. However, hierarchical aggregation still requires fast and stable connections between clients within the same cluster. Consequently, clustering-based solutions struggle to adapt cross-silo FL networks to the heterogeneous and fluctuating conditions of WANs due to the sparsity of silos and their geo-distributed nature. Therefore, *optimizing communication efficiency in cross-silo FL without sacrificing model quality remains an open challenge*.

This paper proposes FedCod, the first application-layer *coded* communication protocol tailored to geo-distributed cross-silo FL, which integrates coding into both download and upload phases to optimize communication efficiency and accelerate FL training. Specifically, FedCod adapts to heterogeneous and fluctuating networks transparently by employing dynamic coding redundancy and coded aggregation strategies.

The coded communication protocol, at the core of FedCod, has been seamlessly integrated into the whole life cycle of cross-silo FL to address the challenges of WAN communications. To mitigate the impact of limited WAN bandwidth between data silos, FedCod encodes the model weights into redundant data blocks, structuring multiple transmission paths

*Corresponding author

979-8-3195-1703-6/26/$31.00

between sources and sinks through client-to-client communication, utilizing the idle bandwidth in cross-silo FL networks. The coding redundancy enables cross-silo FL to tolerate heterogeneous and bottleneck links by adaptively selecting faster links to retrieve encoded data blocks. This mechanism ensures that model weights can be efficiently recovered without being blocked by slow or faulty links. Besides, FedCod allows clients to forward encoded data blocks from their neighbors to the server and pre-aggregate with their own blocks. The client-side pre-aggregation further reduces the upward bandwidth consumption to the server. Meanwhile, FedCod's adaptive redundancy algorithm establishes a minimum level of coding redundancy to ensure resilience against network fluctuations. FedCod gradually reduces the redundancy when performance is stable. When performance fluctuations are detected, FedCod promptly restores high redundancy to maintain efficient communication, effectively avoiding network bottlenecks and fluctuations in cross-silo FL networks.

Our main contributions are as follows:

- We are the first to introduce application-layer *coded* communication protocol tailored to geo-distributed cross-silo FL, which addresses network fluctuations and heterogeneity to enhance collaborative training efficiency from a network perspective.
- We propose FedCod, an application-layer coded communication protocol that leverages adaptive coding redundancy to enhance cross-silo FL communication efficiency while reducing traffic.
- We empirically evaluate FedCod in diverse real-world cross-silo environments, showing that it reduces communication overhead by up to 62% compared to the baseline protocol while preserving FL training performance and optimizing inter-client communication traffic.

II. Background & Motivation

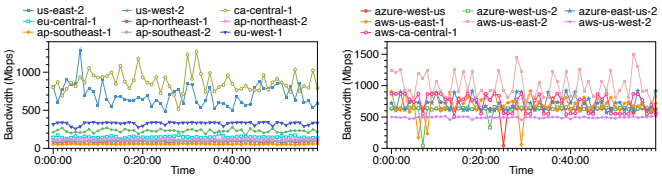
A. Cross-Silo FL

According to FL participant types and distribution, FL can be categorized into *cross-device* and *cross-silo* FL. The cross-device FL scenario involves a large number of clients typically with limited processing power and unreliable communication links [3]. Unlike cross-device FL, cross-silo FL participants are usually organizations (*e.g.*, medical institutions) with sufficient computing resources, connected by WANs, typically housed in geo-distributed data centers [3].

A cross-silo FL system involves a server and n clients.[1] Each client i possesses a local dataset D_i. In each communication round t, the activities of the server and the clients can be divided into the following phases: 1) *Download phase*: The server broadcasts the global model W_t to all clients. The download time for client i is denoted as $T_{\text{download}}(i)$. 2) *Training phase*: Each client i receives the global model W_t as the initial model W^i_{t+1}, and performs several iterations of local training on its local dataset. Client i's training time is defined as $T_{\text{train}}(i)$. 3) *Upload phase*: After local training, each client i uploads its model weights W^i_{t+1} to the server, which has a single sink and multiple sources. The upload time is $T_{\text{upload}}(i)$. 4) *Aggregation phase*: The server aggregates the received local models using a predefined aggregation algorithm. For example, FedAvg [32] and FedProx [2] use a weighted average to update the global model: $W_{t+1} = \sum_i \frac{|D_i|}{\sum_i |D_i|} W^i_{t+1}$. Many other algorithms also aggregate linearly [33], [34].

The total time $T(i)$ from the start of the download phase to the receipt of the local model from client i at the server is given by $T(i) = T_{\text{download}}(i) + T_{\text{train}}(i) + T_{\text{upload}}(i)$. Therefore, the duration of one communication round is $T = \max_i T(i)$, and client i's waiting time is defined as $T_{\text{wait}}(i) = T - T(i)$.

[1]The "server" and "clients" are logical roles in cross-silo FL. The "server" indicates the organization who orchestrates the learning and aggregates weights. The "clients" are organizations performing training on local data.



(a) Global profiling results (b) North America profiling results

Fig. 1. Communication bandwidth profiling results.

B. Measuring Real-world Cross-Silo Networks

In real-world cross-silo FL scenarios, clients experience prolonged communication due to limited and fluctuating WAN bandwidth, even though equipped with high-speed network interfaces and broadband Internet access. To understand and quantify the challenges in cross-silo FL, we measure cross-silo FL networks and evaluate the issues highlighted above: *network heterogeneity* and *bandwidth fluctuations*.

We used `iPerf` to profile the communication bandwidth between the server and clients over an hour for both regional (North America) and global network topologies of cross-silo FL. More experimental details refer to Section V-B1. The results in Figures 1(a) and 1(b) illustrate the challenges of *network heterogeneity* and *bandwidth fluctuations*.

Spatially, client-server bandwidth varies widely from tens of megabits per second to over 1 Gbps, reflecting severe *network heterogeneity* largely driven by geographic distance (*e.g.*, intercontinental links). Such slow links create stragglers that bottleneck synchronous FL. *Temporally*, per-link bandwidth is highly volatile and may drop to near zero, so large model transfers can take minutes and experience multiple bandwidth peaks and valleys, making scheduling difficult.

These observations indicate that cross-silo FL requires communication schemes robust to both heterogeneous and time-varying WAN bandwidth.

C. Motivating Coded Communication

Coding provides space-efficient redundancy, enabling systems to recover from failures and bottlenecks by reconstructing lost data from redundant pieces [35], [36], [37]. As illustrated in Figure 2, we can recover the data $G = [G_1, G_2]$ from the encoded data blocks 1 and 3, even though block 2 is lost.

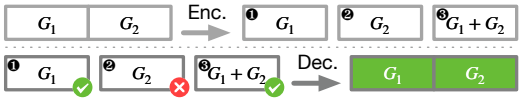
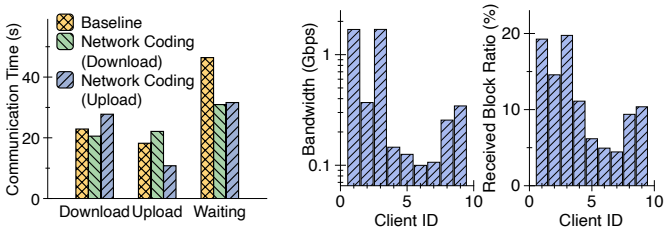


Fig. 2. A simple example for coding.



(a) The communication overhead for baseline and our two adapted network coding communication protocols (one in download phase and the other in upload phase).

(b) Bandwidth and upload workload distribution between different clients and server. (Left) Bandwidth distribution. (Right) Proportion of blocks received from each client during upload.

Fig. 3. Motivating Example.

In cross-silo FL, bandwidth is often underutilized: while some client-server links are bottlenecked, other client-to-client paths remain idle. This creates an opportunity to exploit multi-path transfers, where model blocks can be relayed across clients and recovered from a subset of received blocks via coding, making communication robust to heterogeneous and time-varying WAN bandwidth.

To motivate our design, we run a real-world experiment using random linear network coding (RLNC) [38] in both the global download and local upload phases. As shown in Figure 3(a), RLNC reduces straggler waiting in the download phase by completing clients' downloads more synchronously, and cuts both upload time and waiting time by 30% in the upload phase. Intuitively, coding redundancy lets faster links contribute more blocks without being gated by slower links (Figure 3(b)). Motivated by these results, we design coding strategies tailored to cross-silo FL dataflow to adapt to network heterogeneity and fluctuations.

III. Overview

A. Objectives & Challenges

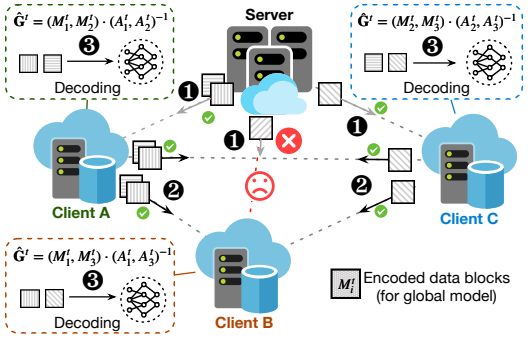
FedCod is designed to achieve the following objectives:

O1: Accelerating FL by optimizing the communication efficiency. FedCod aims to accelerate FL by optimizing communication efficiency with coded communication protocols that fully utilize cross-silo FL networks' available bandwidth.

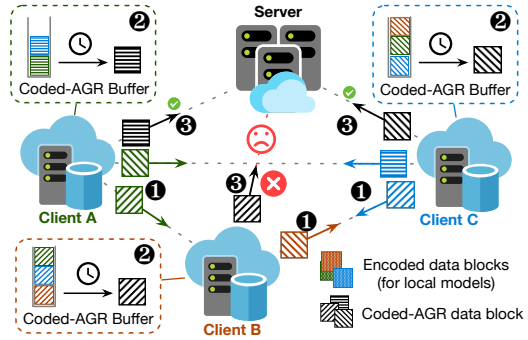
O2: Adapting to fluctuating network conditions. As network conditions across geo-distributed data silos exhibit spatial network heterogeneity and temporal bandwidth fluctuations, FedCod should be able to adapt to varying network conditions by carefully applying coding techniques.

O3: Compatibility with existing FL algorithms. As an application-layer coded communication protocol, the design should be compatible with existing FL algorithms.

To achieve these objectives, we should address the corresponding challenges:



(a) Download



(b) Upload

Fig. 4. FedCod's workflow. (Illustrated with the number of model partitions $k = 2$ as an example.)

First, *designing effective coding strategies for cross-silo FL is non-trivial*. Coding must balance redundancy and latency mitigation: insufficient redundancy reduces robustness under heterogeneity and fluctuations, while excessive redundancy wastes bandwidth and increases overhead. Moreover, centralized FL can bottleneck at the server due to heavy ingress and egress traffic. Thus, FedCod should optimize coding to improve both fault tolerance and communication efficiency while reducing server traffic.

Second, *the variety of network conditions* make it hard to identify optimal transmission paths and balance load across clients. Leveraging idle bandwidth via client-to-client transfers introduces additional paths and can improve efficiency. However, static optimization policies quickly become suboptimal under bandwidth fluctuations, and real-time global profiling is often infeasible, complicating online adaptation.

Lastly, *existing communication-efficient FL schemes are often tightly coupled with specific FL algorithms* (*e.g.*, aggregation strategies or communication frequency), which hinders integration and limits generality. We therefore seek a communication protocol that can be plugged into different cross-silo FL algorithms with minimal assumptions.

B. FedCod's Workflow

FedCod improves communication efficiency in cross-silo FL by deploying coded transfer to the *Download* and *Upload* phases. Both the server and clients perform lightweight cod-

ing in addition to standard transmission, while *Training* and *Aggregation* remain unchanged.

In the *Download* phase (Figure 4(a)), the server encodes model partitions with random coefficients: **Step ❶**: The server sends distinct encoded data blocks to clients; **Step ❷**: Each client forwards received data blocks to its neighboring clients; **Step ❸**: A client decodes the global model once enough encoded data blocks are collected.

The *Training* phase follows the standard procedure in which each client performs local training and updates its model.

In the *Upload* phase (Figure 4(b)), clients encode local updates using a shared coefficient sequence: **Step ❶**: The clients send the encoded data blocks to designated neighbors according to the predefined mapping; **Step ❷**: Each client maintains a Coded-AGR buffer and aggregates the encoded data blocks with the same coefficient vector into an AGR data block; **Step ❸**: The clients send the AGR data blocks to the server, which decodes the aggregated global model.

After *Aggregation*, an adaptive redundancy algorithm updates coding parameters to improve efficiency and traffic.

IV. FedCod's Design

We introduce FedCod, an application-layer coded communication protocol for centralized cross-silo FL to improve communication efficiency. FedCod consists of **coding strategies** and an **adaptive redundancy algorithm**. By applying different coding strategies for distinct data flow characteristics of the download and upload phases, FedCod optimizes bandwidth utilization and accelerates the transmission of model weights through client-to-client communication. Increased coding redundancy improves communication efficiency and system tolerance to network fluctuations, but more redundancy than necessary results in wasted communication traffic. The adaptive redundancy algorithm dynamically adjusts the level of coding redundancy to changing network conditions, balancing increased tolerance with increased communication traffic.

A. Coding for the Download Phase

In the download phase, the server distributes the global model to all clients, which is a single-source content distribution task. The basic server-client communication protocol results in repetitive transmission of identical data from the server to multiple clients, creating a bottleneck at the server due to the high volume of egress traffic.

Server-side Encoding. To reduce the transmission of duplicate data, the server splits the global model into multiple partitions of equal size and encodes these partitions into encoded data blocks before sending them to the clients. Specifically, the global model $\mathbf{G}^t$ is divided into k equal-sized partitions at each communication round, denoted as $\mathbf{G}^t = (G_1^t, G_2^t, \ldots, G_k^t)$, where t indicates the t-th communication round. New encoded blocks $\{M_i^t\}$ are continuously generated by different linear combinations of the model partitions with the random coefficient vector $A_i^t = (\alpha_{i,1}^t, \alpha_{i,2}^t, \ldots, \alpha_{i,k}^t)^T$, *i.e.*,

$$M_i^t = \mathbf{G}^t \cdot A_i^t = \sum_{j=1}^{k} \alpha_{i,j}^t \cdot G_j^t. \tag{1}$$

The server generates the coefficient vector A_i^t and appends it to the block header. It sends encoded data blocks to clients until each can decode the global model. Since coding cost grows with the number of blocks [39], it is necessary to bound the number of partitions k.

Client-side Decoding. According to coding theory, if a client has received k different data blocks $\{M_{i_j}^t\}_{j=1,\ldots,k}$ with the corresponding coefficient vectors $\mathbf{A} = (A_{i_1}^t, A_{i_2}^t, \ldots, A_{i_k}^t)$, these vectors are linearly independent with a high probability close to 1. The client can recover the original k global model partitions with Gaussian elimination as the following equation:

$$\begin{aligned} \hat{G}_i^t &= \sum_{j=1}^{k} \mathbf{A}^{-1}[i][j] \cdot M_{i_j}^t, \text{ for } i = 1, \ldots, k, \\ &\textit{i.e.}, \hat{\mathbf{G}}^t = (M_{i_1}^t, \ldots, M_{i_k}^t) \cdot \mathbf{A}^{-1}. \end{aligned} \tag{2}$$

Client-assisted Forwarding. The randomness of the coding coefficients ensures that any l ($l \leq k$) encoded blocks sent by the server are linearly independent. The clients can leverage the available client-to-client links to forward blocks received from the server to neighboring clients. Therefore, under this strategy, each client only needs to receive any k encoded blocks to achieve the global model, regardless of which communication link they come from. The server significantly reduces egress traffic, and the client benefits from faster communication links while reducing its dependence on a single channel.

Distinct from network coding, we do not perform client-side re-encoding and we do not forward data blocks received from other clients, due to the fact that they would result in linear dependency of the forwarded data blocks [40]. In contrast, our forwarding strategy ensures that there is no duplicate data transmission by linearly independent coding, and eliminates the overhead of client-side re-encoding and memory copying of data blocks, making communication more efficient.

B. Coding for the Upload Phase

In the upload phase, we adopt a coding and forwarding strategy similar to that of the download phase to improve communication efficiency. Each client encodes its local model into different data blocks and sends them to the server and to neighboring clients. The clients then forward the data blocks received from their neighbors to the server. The server decodes and aggregates the local models from the received data blocks.

To efficiently manage the transmission process, each client prioritizes uploading its own data blocks. Specifically, each client maintains two upload queues: *own-queue* and *other-queue*. The *own-queue* is dedicated to the uploading of k encoded data blocks of its local model. The *other-queue* is used for forwarding the encoded data blocks from other clients. If the *own-queue* is not empty, the client prioritizes sending blocks in this queue to the server; otherwise, the client sends data blocks from the *other-queue* to the server in a first-in, first-out (FIFO) order.

C. Coded Aggregation

Unlike the download phase, the upload phase involves multiple sources sending data to a single sink, resulting in

heavier and more complex data flow. The amount of information that the server needs to receive is n times the amount sent during the download phase and grows linearly with the number of clients, which can make the server a bottleneck as n increases. At the same time, the coding redundancy increases exponentially, contributing further to the network load. Therefore, addressing the network issues through coding alone is challenging.

Considering that coding works linearly on model partitions and most existing aggregation algorithms are linear, we design the Coded Aggregation (Coded-AGR) based on the coding strategy for upload in Section IV-B. By aggregating encoded data blocks on the client side, Coded-AGR allows clients' local model partitions to hitchhike on other clients' local model partition uploads, which can reduce bandwidth occupancy during the transmission of data blocks. Benefiting from both coding and on-client aggregation, this approach improves network fluctuation tolerance and communication traffic savings.

During the upload phase, each client encodes its local model into different data blocks and transmits them to other clients. Then, on the client side, data blocks with the same coefficient vector are aggregated into an AGR data block that is forwarded to the server. Instead of reconstructing all individual local models, the server can reconstruct an aggregated global model from the AGR data blocks.

Specifically, all clients generate the same sequence of coefficient vectors required for coding as agreed upon in advance, *e.g.*, based on the Cauchy matrix [41], [42]. For any two clients i_1 and i_2, the j-th generated coefficient vectors $(\alpha_{j,1}^{t,i_1}, \ldots, \alpha_{j,k}^{t,i_1})$ are equal to $(\alpha_{j,1}^{t,i_2}, \ldots, \alpha_{j,k}^{t,i_2})$. For simplicity, we denote the j-th coefficient vector as $\alpha_j^t = (\alpha_{j,1}^t, \ldots, \alpha_{j,k}^t)$. Furthermore, they use the same mapping sequence $h(\cdot)$ to transmit the encoded data blocks to different clients, *i.e.*, the j-th data block of the client i, $M_j^{t,i}$, is sent to the client $h(j)$. For example, we define $h(j) = j \mod n$ as the default mapping.

When client $h(j)$ receives the data blocks $\{M_j^{t,i}\}$ from another client i, it aggregates them and its own j-th data block into a new data block $\tilde{M}_j^t = \sum_{\text{received } i} M_j^{t,i}$. For client $h(j)$, it can choose to upload the data block $\tilde{M}_j^t$ once a predefined time window has elapsed (*non-wait mode*), or it can upload the AGR data block after all the j-th blocks from all clients have been received and aggregated (*wait mode*).

Wait mode coded aggregation is theoretically more communication-efficient than *non-wait mode*; we omit the proof due to space constraints. Unless stated otherwise, we use coded aggregation to refer to *wait mode*.

D. Adaptive Redundancy Algorithm

The improvement in communication efficiency through coding strategies relies on coding redundancy. This redundancy enables clients with faster communication links to the server to assist neighboring clients in transmission, providing the system with tolerance to network fluctuations and link failures. Inadequate redundancy can lead to less efficient communication, while excessive redundancy can lead to wasted communication traffic, which has a significant impact on client costs. To address this issue, we design the adaptive redundancy algorithm, which dynamically adjusts the coding redundancy to adapt to varying network conditions.

Overall, this algorithm responds to communication time variations caused by network fluctuations and sets a lower bound on redundancy to provide sufficient tolerance for common network fluctuations. The system starts with a high redundancy state, gradually reduces the redundancy if the communication time does not increase, and quickly restores high redundancy if performance fluctuations are detected.

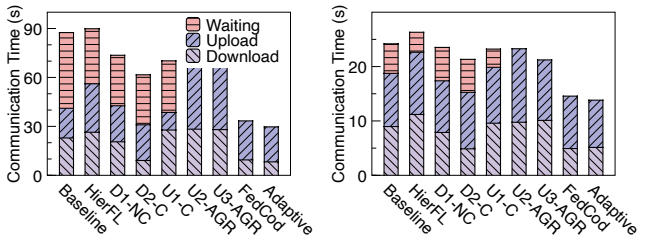
Our adaptive redundancy algorithm consists of three parts: *cold start*, *redundancy reduction*, and *rapid recovery*. *1) Cold start*: We define the number of encoded data blocks exceeding k as the redundancy number r, and the coding redundancy of the system is defined as r/k. r is initialized to a large number, indicating a high tolerance for network fluctuations. *2) Redundancy reduction*: The variables t_{last} and t_{cur} denote the communication duration of the last and the current communication round, respectively. r is gradually reduced over time to reduce the redundancy of the communication volume. This reduction continues when the current communication duration t_{cur} is not larger than the last communication duration t_{last}, indicating a decrease in redundancy demand. We also set a lower bound r_{lb} for r and adjust it according to the network conditions. To make the system less sensitive to small network fluctuations, we introduce a scaling factor $\lambda > 1$ and the system compares t_{cur} and $t_{\text{last}} * \lambda$ at the time of judgement. *3) Rapid recovery*: If some links fail or bandwidth fluctuates, *i.e.*, t_{cur} exceeds $t_{\text{last}} * \lambda$, r should be increased proportionally to ensure system resilience. The system should also increase the lower bound r_{lb} because there is at least one communication path that is getting worse. The system will continue to adjust upwards r in multiple rounds until the optimization of the system communication time is no longer apparent, *i.e.*, $t_{\text{cur}} \geq t_{\text{last}}/\lambda$. In addition, if there are no further network fluctuations for a period of time, the system will decrease the lower bound of redundancy r_{lb}.

In essence, the algorithm dynamically adapts redundancy levels by starting with a high redundancy, gradually decreasing it to mitigate redundancy, and then increasing redundancy in response to network fluctuations. This iterative adjustment mechanism allows the system to optimize redundancy levels in accordance with evolving network conditions, thereby increasing overall efficiency and robustness.

It should be noted that adaptive redundancy is unnecessary in the download phase because transmissions are sequential per client and can be halted as soon as local decoding is possible, thus avoiding waste. In the upload phase, however, redundancy should be pre-arranged, making adaptive control essential.

V. Evaluation

This section answers the following questions: 1) What is the improvement in communication efficiency of the FL system using FedCod compared to the basic server-client communication protocol? 2) What is the impact of FedCod



(a) In the global topology (b) In the North America topology

Fig. 5. The communication time of different protocols. Baseline: the basic server-client communication protocol; HierFL: the hierarchical FL communication protocol; D1-NC: network coding in the download phase; D2-C: our coding strategy only in the download phase; U1-C: our coding strategy only in the upload phase; U2-AGR: non-wait mode Coded-AGR in the upload phase; U3-AGR: wait mode Coded-AGR in the upload phase; FedCod: FEDCOD with static redundancy; Adaptive: FEDCOD with adaptive redundancy.

on the communication traffic? 3) Does FEDCOD affect the learning convergence? FEDCOD achieves up to 62% reduction in communication time compared to the baseline and has the same convergence of the global model as the baseline.

A. Experimental Setup

We implement FL in PyTorch and use gRPC over TCP for communication. For silo locations, we consider two topologies: Global and North America:

Global topology. We evaluate on Amazon Web Service (AWS) platform across 10 regions using one `p3.8xlarge` EC2 instance per region: `us-east-1` (N. Virginia), `us-east-2` (Ohio), `us-west-2` (Oregon), `ca-central-1` (Canada), `ap-northeast-1` (Tokyo), `ap-northeast-2` (Seoul), `ap-southeast-1` (Singapore), `ap-southeast-2` (Sydney), `eu-central-1` (Frankfurt), and `eu-west-1` (Ireland). Each instance has 32 vCPUs and up to 10 Gbps network bandwidth.

North America topology. We deploy on 8 North America regions: 4 AWS EC2 regions (`us-east-1`, `us-east-2`, `us-west-2`, `ca-central-1`) and 4 Azure regions (`central-us`, `west-us`, `west-us-2`, `east-us-2`). To reduce cost and focus on communication optimization, we sample client training time from an empirical distribution instead of performing GPU training. On AWS, we use `m5.8xlarge` instances (32 vCPUs, comparable network interface). On Azure, we use `Standard_D32a_v4` instances (32 vCPUs, up to 16 Gbps), providing similar compute and network capacity.

In the global topology, we use AWS `us-east-1` as the server; the remaining instances are clients. In the North America topology, we use Azure `central-us` as the server and the rest as clients.[2] We run 10 FL rounds per protocol and report the average phase time (Section II-A). Unless otherwise specified, the reported communication time is the end-to-end wall-clock time measured by the FL runtime, including data transfer as well as the additional protocol operations

[2]We choose a geographically central region, following the setup in [4].

TABLE I
THE AVERAGE SERVER TRAFFIC (UNIT: MBYTES)

Global			North America		
Protocol	**Ingress**	**Egress**	**Protocol**	**Ingress**	**Egress**
Baseline	4003.697	4003.697	Baseline	3113.987	3113.987
D1-NC	4003.697	2342.916	D1-NC	3113.987	1207.469
D2-C	4003.697	1334.572	D2-C	3113.987	1487.093
U1-C	7612.005	4003.697	U1-C	4639.221	3113.987
U2-AGR	8037.091	4003.697	U2-AGR	3609.695	3113.987
U3-AGR	444.857	4003.697	U3-AGR	444.857	3113.987
FEDCOD	**444.857**	**1314.801**	**FEDCOD**	**444.857**	**1188.774**

introduced by FEDCOD, such as encoding, decoding, forwarding, and Coded-AGR. We also measure average ingress/egress traffic at the server and clients to quantify communication cost. In addition, we conduct numerical studies on the number of model partitions and coding redundancy, and run conformance experiments on a local cluster to verify learning convergence. We use ResNet152 (60M parameters; ∼440 MB packed weights) trained on federated datasets generated from the CIFAR-10 following [43]. Unless specified, we set the number of partitions $k = n$ and redundancy to 100%.

B. Experimental Results

1) Effectiveness of Coding Strategies: First, we show that Hierarchical FL (HierFL) [30] is not suitable for the geo-distributed cross-silo FL scenario. We group data silos based on their geographical location and communication status and perform hierarchical aggregation within the group. In each group, the data silo with the fastest communication speed to the server is selected as the center for hierarchical aggregation. For example, in the global topology, the silos are divided into three groups, North America, Asia, and Europe. The centers of hierarchical aggregation are `us-east-2`, `ap-northeast-1` and `eu-central-1`. Figure 5 shows that HierFL is even worse than the baseline. With the more detailed composition of communication time per client in Figure 6, we can see that the communication time between clients within each group is non-negligible. As a result, forwarding most clients' data through the hierarchical aggregation center is less efficient than communicating directly with the server.

To demonstrate the effectiveness of each component of our FEDCOD, we apply our designed coding strategies separately to the download and upload phases, and finally in combination, within the global topology. The adaptive redundancy algorithm of FEDCOD is discussed in Section V-B2; for now, we use static redundancy (with default redundancy 100%).

Figure 5(a) shows that applying existing network coding in the download phase (D1-NC) does not significantly reduce the average download time, but it does reduce the average waiting time by about 33%. According to Figure 6, D1-NC makes the end of each client download phase closer. As the analysis in Section IV-A shows, only the forwarding with the encoded blocks from the server is always valid, and the forwarding with the encoded blocks from other clients sometimes wastes the bandwidth. We randomly select a communication round and count the validity of the encoded blocks received by each client. Figure 7 shows that up to 20% of the encoded blocks

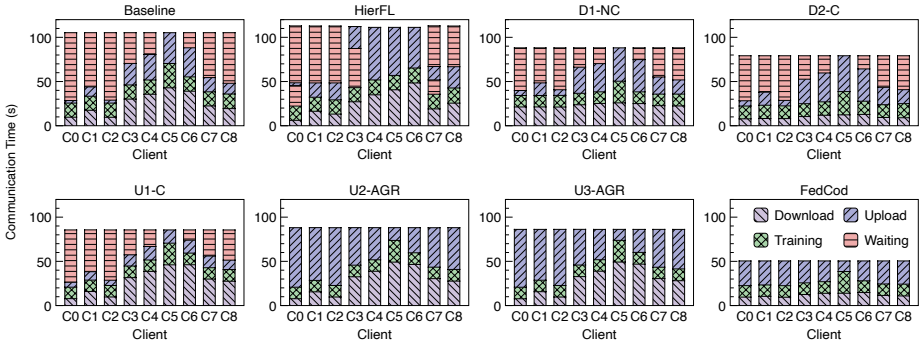


Fig. 6. The communication time for each client of different communication protocols.

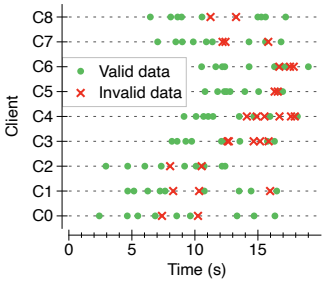


Fig. 7. Validity of encoded data blocks received on each client under network coding. The x-axis shows time since the server distributes the global model.

received by the client are invalid, which not only wastes traffic but also delays the transmission of other valid encoded blocks. The benefits brought by coding and forwarding are offset by this invalid data transmission. Therefore, the average download time of D1-NC is quite close to the baseline time.

When using our coding strategy in the download phase (D2-C), the client's forwarding efficiently uses the idle bandwidth through the client-to-client communication. The average download time is reduced by 60% and the average waiting time is reduced by 33% compared to the baseline. According to Figure 6, the download speeds of all clients have benefited from coding and the effective forwarding, especially the slower clients such as C5. In addition, Table I shows that server egress traffic is saved by 41% (D1-NC) and 67% (D2-C), respectively. Due to the coding mechanism, the data sent by the server is not duplicated, and the clients have more chances to get the global model partitions from their neighbors. Since each time forwarding is effective for D2-C, there are significant egress traffic savings in D2-C compared to D1-NC.

When our coding strategy is applied in the upload phase (U1-C), the average upload time is reduced by about 40% and the average waiting time is reduced by about 32%. When applying coded aggregation (U2-AGR/U3-AGR), the server decodes an aggregated global model from received blocks rather than per-client models; thus, their upload-phase time should be compared against the sum of upload and waiting time in other schemes. U2-AGR and U3-AGR achieve similar upload-phase communication time to U1-C, which is about 62%-69% of the communication time in the upload phase of the baseline. However, the ingress traffic of the server with U1-C and U2-AGR is relatively high because the algorithms require transmission redundancy for communication efficiency. The wait mode Coded-AGR (U3-AGR) can significantly reduce the ingress traffic through the hierarchical aggregation on encoded data blocks. Table I shows that the cost of accelerated communication in the upload phase of U1-C and U2-AGR is nearly twice as much server ingress traffic as in the baseline. The ingress traffic of U3-AGR is only about 11% of the baseline. With the full application of our coding strategies, FedCod reduces the average download time by 59% and the total communication time by 62% compared to the baseline.

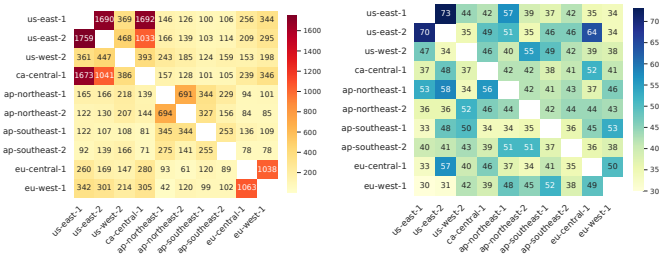
In the North America topology, which is quite different from the global topology, the system has less communication heterogeneity as shown in Figure 1(b). Therefore, the download time, upload time, and waiting time of the baseline are small, which are only about 9s, 9.8s, and 5.4s, respectively. In this case, the advantage of existing network coding in the download phase (D1-NC) over the baseline is even smaller. Figure 5(b) shows that our coding strategy in the download phase (D2-C) still achieves a 46% improvement in the download time. In the upload phase, our coding strategy with wait mode Coded-AGR (U3-AGR) reduces the upload communication time by 27% and decreases server ingress traffic to just 14% of the baseline. With the full application of our coding strategies, FedCod reduces the total communication time by about 40%.

TABLE II
THE AVERAGE TRAFFIC (UNIT: MBYTES)

Topology	Role	Protocol	Ingress	Egress
Global	Server	Static	444.857	1314.801
		Adaptive	444.857	1448.258 ↑
	Client	Static	1656.407	1693.754
		Adaptive	1564.140 ↓	1586.658 ↓
North America	Server	Static	346.000	1235.715
		Adaptive	346.000	1163.819 ↓
	Client	Static	1254.074	1501.217
		Adaptive	942.353 ↓	1124.019 ↓

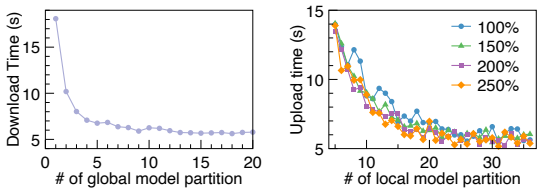
2) Effectiveness of Adaptive Redundancy Algorithm: To evaluate adaptive redundancy algorithm, we compare FedCod under static vs. adaptive redundancy in terms of communication time and inter-client traffic. We set $\lambda = 1.2$ by default.

For the global topology, Figure 5(a) shows that the communication time of FedCod with the adaptive redundancy is 11% lower than the time of FedCod with static redundancy. In addition, the adaptive redundancy algorithm reduces communication traffic between clients. Table II presents the average ingress and egress traffic for all clients, as well as the individual ingress and egress traffic for servers, under both static and dynamic redundancy. The results indicate that the dynamic mechanism reduces inter-client communication traffic by 6%. For the North America topology, Figure 5(b) shows that the communication time is comparable between the two schemes due to lower network heterogeneity and fewer fluctuations. The adaptive redundancy algorithm reduces the redundancy level dynamically, achieving up to a 25% decrease in inter-client communication traffic.



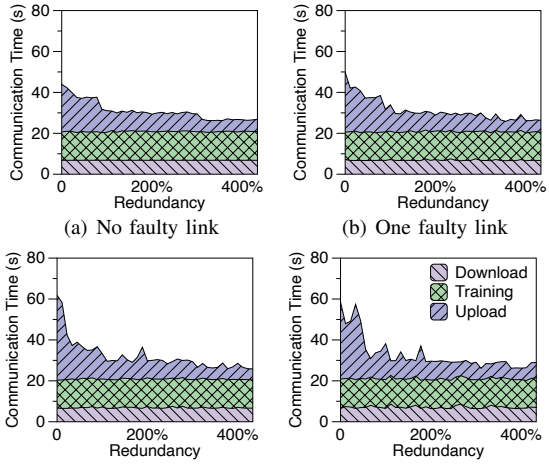
(a) Bandwidth Mean (Unit: Mbps) (b) Variance (Unit: Mbps^2)

Fig. 8. The real-world distribution of the communication bandwidth between each pair of silos in the global topology.



(a) Download k (b) Upload k

Fig. 9. The impact of the number of model partitions k.



(a) No faulty link (b) One faulty link

(c) Two faulty links (d) Three faulty links

Fig. 10. The communication time for different redundancy and faulty links in the global topology.

3) Numerical Experiments: We conduct numerical experiments to study the impact of the number of model partitions and how redundancy affects tolerance to network fluctuations. We simulate FedCod using the measured inter-silo bandwidth in the global topology (Figure 8, mean and variance), and incorporate empirical distributions of training and coding time.

Impact of the number of model partitions k. In coding strategies, the number of model partitions k is an important factor that affects the communication time. We study the effect of k in the download and upload phases separately.

For the download phase, Figure 9(a) shows that when $k = 1$, the client can forward the data to its neighbors only after receiving the full global model and can hardly benefit from the coding, so the average download time is very close to the baseline. As k grows, the model is partitioned at a finer granularity. The model partitions have more chances to be encoded and forwarded to the clients by the neighboring clients with good communication status, so the download time decreases.

TABLE III
COMMUNICATION TIME AND SPEEDUP OF DIFFERENT PROTOCOLS

Protocol	Communication time (s)	Speedup
Baseline	83.18	1×
CREW [44]	39.04	2.13×
FedCod	**32.68**	**2.54**×

TABLE IV
THE TESTING ACCURACY OF DIFFERENT COMMUNICATION PROTOCOLS IN DIFFERENT TRAINING STAGES

Protocol	Epoch=20	Epoch=40	Converge
Baseline	0.70	0.78	0.80
HierFL	0.70	0.79	0.80
D1-NC	0.73	0.79	0.80
D2-C	0.72	0.79	0.80
U1-C	0.70	0.79	0.80
U2-AGR	0.70	0.79	0.80
U3-AGR	0.70	0.79	0.80
FedCod	0.70	0.79	0.80
Adaptive	0.70	0.78	0.80

For the upload phase, we experiment with the impact of the number of local model partitions for four different redundancy scenarios (100%, 150%, 200%, and 250%). Figure 9(b) shows that the upload time decreases as k increases in four scenarios. The reason is that large k gives the model partitions more opportunity to be aggregated and forwarded to the server by clients with fast transmission speeds. For both two phases, due to the inherent coding time of each model partition, the tendency to decrease stops when k exceeds a threshold. Therefore, in practice, we should choose the appropriate k based on the number of clients and the coding overhead.

Impact of the coding redundancy. Figure 1 shows that the real-world bandwidth fluctuations can lead to link failures (bandwidth at very low levels). We conduct experiments for the global topology under different numbers of faulty links to explore the relationship between redundancy and system tolerance. Figure 10 shows that the performance improvement of increasing redundancy is significant even in the absence of link failures. This is because redundancy improves the utilization of idle bandwidth through client-to-client communication, and more data reaches the server through faster clients. As the number of faulty links increases, the system requires more and more redundancy to cope with the faulty links to maintain a stable communication efficiency.

4) Comparison with GDML communication: We compare FedCod with CREW [44] as a contextual reference, since CREW targets geo-distributed machine learning with a different synchronization model and is not a strict drop-in baseline for centralized cross-silo FL. Under the same global topology with `m5.8xlarge` instances, Table III shows that CREW achieves a 2.13× speedup over the baseline, while FedCod achieves a 2.54× speedup. This is because CREW relies on linear programming over profiled bandwidth status, whereas FedCod uses coding to opportunistically adapt to bandwidth heterogeneity and fluctuations.

5) Conformance Experiments: Our proposed algorithms for cross-silo FL, including the coding strategies and the adaptive redundancy algorithm, are lossless communication protocols.

Therefore, they do not affect the learning convergence of the global model. Table IV shows that our coding strategies in both the download and upload phases do not affect the convergence of the global model. The global models trained in the systems with coded aggregation and adaptive redundancy algorithm can also converge to the test accuracy with the same convergence rate and the same accuracy level.

VI. Related Work

A. Communication-Efficient FL

Communication-efficient FL methods fall into two categories: *algorithmic solutions* that trade convergence or accuracy for communication efficiency, and *network optimizations* decoupled from FL algorithms.

Algorithmic solutions. Most existing communication-efficient FL methods attempt to address the communication problem from an algorithmic perspective. To improve the communication efficiency, *they rely on specific FL setups and cannot be applied to generic FL algorithms*. Some previous work increases the number of local training iterations between communication rounds and decreases the frequency of communication [23], [32], [45]. While lower communication frequencies reduce the amount of data exchanged, they are more likely to introduce model drift [25]. Quantization and compression techniques in FL also reduce the amount of communication per round [20], [21], but the loss of information due to lossy compression can affect convergence [26], [27]. Considering that communication delays are usually caused by unstable connections between server and clients, some studies propose one-shot [46] or decentralized FL [4], [47], which modify the centralized architecture and aggregation algorithm. However, modifications in the aggregation algorithms always affect the learning convergence [28].

Network optimizations. From a network perspective, only a few studies use hierarchical aggregation to address the system heterogeneity and save communication resources [29], [30], [31]. They divide the clients into clusters based on the network topology and geographic proximity, and perform early aggregation in the cluster. The early aggregated models are uploaded from the clusters to the server for final aggregation. Clients in the same cluster often have more stable network connections to each other, and early aggregation in the group can reduce bandwidth consumption, speeding up the upload process. This paradigm is more appropriate for cross-device scenarios where most clients are in the same Local Area Network (LAN) or base station coverage area. While in the cross-silo scenario, there are fewer opportunities to leverage this type of "locality" due to the geo-distributed nature. Instead, if the transmission speed between the clients performing early aggregation is not fast enough, the efficiency will be worse than traditional direct server-to-client communication. In contrast, FedCod uses coding strategies to better adapt to cross-silo FL networks in WAN environments, achieving more efficient communication with the same model quality.

B. Network Coding

Network coding improves multicast throughput and robustness by coding packets at intermediate nodes [48], and has been widely used in peer-to-peer content distribution [49]. Random linear network coding uses random coefficients to mitigate packet loss and volatility in such networks [38], [39], [50]. Some recent attempts apply coding ideas to FL but consider limited system settings (*e.g.*, few clients or specific topologies) and do not generalize to practical multi-client cross-silo deployments [51]. Our work builds a general protocol that integrates coding into cross-silo FL dataflow.

C. Coded Computing

Unlike communication redundancy (*e.g.*, network coding in FL), coded computing introduces *computational* redundancy to mitigate stragglers [52], [53]. These methods typically code matrix multiplications across clients, requiring data overlap or scaling poorly to large neural networks. In contrast, we add *communication* redundancy to exploit idle bandwidth in the system, remaining decoupled from the FL algorithms and transparent to the learning process.

VII. Conclusion

This paper proposes FedCod, a new application layer communication protocol designed to improve communication efficiency in cross-silo FL. By using coding strategies specific to the different characteristics of the data flow of the upload and download phases, FedCod makes efficient use of the available bandwidth and accelerates the transfer of model weights through client-to-client communication. Furthermore, the adaptive redundancy algorithm in FedCod dynamically balances redundancy levels in response to network fluctuations, ensuring system resilience while minimizing unnecessary communication traffic. Extensive experiments demonstrate that FedCod significantly reduces communication time without compromising learning convergence.

Acknowledgments

This work was supported by the National Key R&D Program of China (2022YFB4402102), National Natural Science Foundation of China (NO. 62472284), and Shanghai Key Laboratory of Scalable Computing and Systems. The work of Hao Wang was supported in part by NSF 2534286, 2523997, and 2315612.

References

[1] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated Optimization: Distributed Optimization Beyond the Datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[2] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *MLSys*, 2020.

[3] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and Open Problems in Federated Learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[4] O. Marfoq, C. Xu, G. Neglia, and R. Vidal, "Throughput-Optimal Topology Design for Cross-Silo Federated Learning," in *NeurIPS*, 2020.

[5] M. Oldenhof, G. Ács, B. Pejó, A. Schuffenhauer, N. Holway, N. Sturm, A. Dieckmann, O. Fortmeier, E. Boniface, C. Mayer *et al.*, "Industry-Scale Orchestrated Federated Learning for Drug Discovery," in *AAAI*, 2023.

[6] S. Silva, B. A. Gutman, E. Romero, P. M. Thompson, A. Altmann, and M. Lorenzi, "Federated Learning in Distributed Medical Databases: Meta-analysis of Large-Scale Subcortical Brain Data," in *ISBI*, 2019.

[7] P. Courtiol, C. Maussion, M. Moarii, E. Pronier, S. Pilcer, M. Sefta, P. Manceron, S. Toldo, M. Zaslavskiy, N. Le Stang *et al.*, "Deep Learning-Based Classification of Mesothelioma Improves Prediction of Patient Outcome," *Nature Medicine*, vol. 25, no. 10, pp. 1519–1525, 2019.

[8] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection," *Journal of Machine Learning Research*, 2021.

[9] T. Madiega, "Artificial Intelligence Act," *European Parliament: European Parliamentary Research Service*, 2021.

[10] H. Woisetschläger, A. Erben, B. Marino, S. Wang, N. D. Lane, R. Mayer, and H.-A. Jacobsen, "Federated Learning Priorities Under the European Union Artificial Intelligence Act," *arXiv preprint arXiv:2402.05968*, 2024.

[11] R. Ye, W. Wang, J. Chai, D. Li, Z. Li, Y. Xu, Y. Du, Y. Wang, and S. Chen, "OpenFedLLM: Training Large Language Models on Decentralized Private Data via Federated Learning," in *SIGKDD*, 2024.

[12] T. Fan, Y. Kang, G. Ma, W. Chen, W. Wei, L. Fan, and Q. Yang, "FATE-LLM: A Industrial Grade Federated Learning Framework for Large Language Models," *arXiv preprint arXiv:2310.10049*, 2023.

[13] J. Zhang, S. Vahidian, M. Kuo, C. Li, R. Zhang, T. Yu, G. Wang, and Y. Chen, "Towards Building the FederatedGPT: Federated Instruction Tuning," in *ICASSP*, 2024.

[14] W. Kuang, B. Qian, Z. Li, D. Chen, D. Gao, X. Pan, Y. Xie, Y. Li, B. Ding, and J. Zhou, "FederatedScope-LLM: A Comprehensive Package for Fine-Tuning Large Language Models in Federated Learning," in *SIGKDD*, 2024.

[15] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds," in *NSDI*, 2017.

[16] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[17] M. Ye, X. Fang, B. Du, P. C. Yuen, and D. Tao, "Heterogeneous Federated Learning: State-of-the-Art and Research Challenges," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–44, 2023.

[18] A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani *et al.*, "Annulus: A Dual Congestion Control Loop for Datacenter and Wan Traffic Aggregates," in *SIGCOMM*, 2020.

[19] Z. Wang, Z. Li, G. Liu, Y. Chen, Q. Wu, and G. Cheng, "Examination of WAN Traffic Characteristics in a Large-Scale Data Center Network," in *IMC*, 2021.

[20] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "FetchSGD: Communication-Efficient Federated Learning With Sketching," in *ICML*, 2020.

[21] R. Dorfman, S. Vargaftik, Y. Ben-Itzhak, and K. Y. Levy, "DoCoFL: Downlink Compression for Cross-Device Federated Learning," in *ICML*, 2023.

[22] Z. Li, W. Feng, W. Cai, H. Yu, L. Luo, G. Sun, H. Du, and D. Niyato, "Accelerating Geo-distributed Machine Learning with Network-aware Adaptive Tree and Auxiliary Route," *IEEE/ACM Transactions on Networking*, 2024.

[23] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-Effective Federated Learning Design," in *INFOCOM*, 2021.

[24] X. Ouyang, Z. Xie, J. Zhou, G. Xing, and J. Huang, "ClusterFL: A Clustering-based Federated Learning System for Human Activity Recognition," *ACM Transactions on Sensor Networks*, vol. 19, no. 1, pp. 1–32, 2022.

[25] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning," in *ICML*, 2020.

[26] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Hierarchical Federated Learning With Quantization: Convergence Analysis and System Design," *IEEE Transactions on Wireless Communications*, vol. 22, no. 1, pp. 2–18, 2022.

[27] H. Sun, H. Tian, W. Ni, and J. Zheng, "On the Convergence of Hierarchical Federated Learning with Gradient Quantization and Imperfect Transmission," in *ICASSP*, 2024.

[28] Y. Sun, L. Shen, and D. Tao, "Which Mode Is Better for Federated Learning? Centralized or Decentralized," *arXiv preprint arXiv:2310.03461*, 2023.

[29] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin, "Hierarchical Federated Learning Across Heterogeneous Cellular Networks," in *ICASSP*, 2020.

[30] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao, and Y. Zhao, "Resource-Efficient Federated Learning With Hierarchical Aggregation in Edge Computing," in *INFOCOM*, 2021.

[31] Z. Xu, D. Zhao, W. Liang, O. F. Rana, P. Zhou, M. Li, W. Xu, H. Li, and Q. Xia, "HierFedML: Aggregator Placement and UE Assignment for Hierarchical Federated Learning in Mobile Edge Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 328–345, 2022.

[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks From Decentralized Data," in *AISTATS*, 2017.

[33] Z. Li, T. Lin, X. Shang, and C. Wu, "Revisiting Weighted Aggregation in Federated Learning With Neural Networks," in *ICML*, 2023.

[34] Y. Deng, F. Lyu, J. Ren, Y.-C. Chen, P. Yang, Y. Zhou, and Y. Zhang, "Improving Federated Learning With Quality-Aware User Incentive and Auto-Weighted Model Aggregation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4515–4529, 2022.

[35] C. Wu and B. Li, "Echelon: Peer-to-Peer Network Diagnosis With Network Coding," in *IWQoS*, 2006.

[36] C. Feng and B. Li, "On Large-Scale Peer-to-Peer Streaming Systems With Network Coding," in *ACM MM*, 2008.

[37] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *ATC*, 2012.

[38] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[39] B. Li and D. Niu, "Random Network Coding in Peer-to-Peer Networks: From Theory to Practice," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 513–523, 2011.

[40] M. Wang and B. Li, "How Practical Is Network Coding?" in *IWQoS*, 2006.

[41] A. L. Cauchy, *Exercices d'analyse et de physique mathematique: 1*. Bachelier, imprimeur-libraire, 1840, vol. 1.

[42] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-tolerant Network Storage Applications," in *NCA*, 2006.

[43] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu, "FedLab: A Flexible Federated Learning Framework," *Journal of Machine Learning Research*, vol. 24, no. 100, pp. 1–7, 2023.

[44] S. Luo, R. Wang, K. Li, and H. Xing, "Efficient Cross-Cloud Partial Reduce with CREW," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[45] S. U. Stich, "Local SGD Converges Fast and Communicates Little," in *ICLR*, 2019.

[46] J. Zhang, C. Chen, B. Li, L. Lyu, S. Wu, S. Ding, C. Shen, and C. Wu, "DENSE: Data-Free One-shot Federated Learning," in *NeurIPS*, 2022.

[47] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D^2: Decentralized Training Over Decentralized Data," in *ICML*, 2018.

[48] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[49] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," in *INFOCOM*, 2005.

[50] S. Deb, M. Médard, and C. Choute, "Algebraic Gossip: A Network Coding Approach to Optimal Multiple Rumor Mongering," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2486–2507, 2006.

[51] L. Kong, H. Tao, J. Wang, Z. Huang, and J. Xiao, "Network Coding for Federated Learning Systems," in *ICONIP*, 2020.

[52] M. Ye and E. Abbe, "Communication-Computation Efficient Gradient Coding," in *ICLR*, 2018.

[53] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "SlimFL: Federated learning with superposition coding over slimmable neural networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 2499–2514, 2023.